Data Placement and Query Processing Based on RPE Parallelisms

Yaxin Yu, Guoren Wang, Ge Yu, Gang Wu, Junan Hu, Nan Tang
Department of Computer Science and Engineering
Northeastern University, Shenyang 110004, China
yuyx@mail.neu.edu.cn

Abstract

The basic idea behind parallel database systems is to perform operations in parallel to reduce the response time and improve the system throughput. Data placement is a key factor on the performance of parallel database systems. This paper proposes two data partition strategies to decluster XML documents with very large size, *Path Schema based Path Instance Balancing (PSPIB)* strategy, in which all path instances with the same path schema in a data tree are declustered evenly over all sites, and *Node Schema based Node Round-Robin (NSNRR)* strategy, in which all node objects with the same node schema in a data tree are declustered over all sites in a round-robin way. Accordingly, two query processing algorithms are proposed based on the two partition methods, *Parallel Path Merge (PPM)* algorithm and *Parallel Pipelining Path Join (PPPJ)* algorithm. The performance analysis and evaluation on the two data placement strategies and corresponding query processing algorithms are given in this paper.

1. Introduction

Recently, XML has been becoming a dominant standard for information representation and data exchange over the Web [1] and a lot of standards have been being developed for XML representation of data for a variety of specialized applications ranging from business applications to scientific applications. Therefore, XML is considered one of the most promising ways to represent semi-structured data, which is ubiquitous in large volumes from diverse data sources and applications on the web [2]. As a result, issues related to XML data, especially storage and query processing have been becoming hot research topics.

So far, some prototype systems such as Lore [3], Rufus [13], Tamino [14] and so on have been proposed to store and process XML data in uniprocessor environments. However, the performance of these systems is not satisfiable as the size of XML documents to be handled is becoming larger. Therefore, how to store XML documents with large size and how to query them have drawn a lot of attention from both academic community and industrial community. It is no doubt that parallelism should be one of promising ways to solve this problem.

Yuanling Zhu, Kevin Lü *et al* [11] [12] proposed an efficient data placement strategy for XML data. The new strategy is based on a multilevel graph partitioning algorithm with the consideration of the unique features of XML documents and query distributions. But they do not consider the impacts that the features of XML query language posed on data placements. One of common features of many XML query languages, such as Quilt [4], XPath [5], XQuery [6], is the use of regular path expressions (RPEs). Therefore, efficiently dealing with RPE queries has important impact on the performance of XML databases. In this paper, we discuss data placement strategies based on the features of RPE parallelism.

Based on three kinds of traditional parallelisms, i.e., *intra-operator*, *independent* and *pipelining* parallelisms [7] and the features of RPE itself, we propose three kinds

of RPE-based parallelisms to efficiently partition XML documents with large size and query processing on them in this paper, *intra-RPE*, *inter-RPE* and *inter-query* parallelisms. The *intra-RPE* parallelism is how to parallelize an RPE in a multiprocessor system while the *inter-RPE* parallelism is to execute multiple RPEs simultaneously over all sites. The *inter-query* parallelism is to exploit multiple queries simultaneously in multiprocessor systems, which is the same as traditional *inter-query* parallelism. So we focus on the *intra-RPE* and *inter-RPE* parallelisms in this paper. The *intra-RPE* parallelism can be divided further into two types of parallelisms, *inter-SPE* and *intra-SPE* parallelisms since the *inter-RPE* parallelism benefits from the rewritten mechanism for RPE queries, which transforms an RPE into multiple different simple path expressions (SPE) connected by parent-child operators "/". With the *inter-SPE* parallelism multiple SPEs are executed in parallel and a SPE is the atomic parallel execution unit. Similarly, with the *intra-SPE* parallelism multiple path steps of a SPE are executed in parallel and a path step is the atomic parallel execution unit. Based on the *inter-SPE* parallelism, *Path Schema based Path Instance Balancing (PSPIB)* strategy is proposed. An RPE is rewritten to multiple different simple path expressions, each of which is subjected to a certain path schema or part of the path schema. The set of all path instances of each path schema is declustered as evenly as possible over all sites to minimize inter-site references by assigning the traversals related to the same RPE query to the same site and to maximize the system throughput by distributing evenly the workload over all sites. Based on the *intra-SPE* parallelism, *Node Schema based Node Round-Robin (NSNRR)* strategy is proposed. It assigns the element nodes with same node schema to the same site to minimize the query cost and assigns the element nodes with different node schema to different sites in a round-robin way to maximize the system throughput. Just mentioned before, a simple path expression can be divided further into multiple path step operations connected by "/" operator, each of which is subjected to a certain node schema. These path step operations are joined together by the traditional pipelining parallelism to get the final result.

We propose two parallel query algorithms based on the *independent* and *pipelining* parallelisms, *Parallel Path Merge (PPM)* and *Parallel Pipelining Path Join (PPPJ)*. The *PPM* algorithm evaluates an RPE query on a partition of the XML data tree in each site independently and merges the results related to the same RPE query. The *PPPJ* algorithm executes a step of an RPE in each site and joins them in a pipelining way. But due to the limitation of paper space, we don't discuss them here in detail.

The main contributions of this paper are summarized: (1) several kinds of parallelisms are proposed, including *intra-RPE*, *inter-RPE*, *intra-SPE* and *inter-SPE* parallelisms; (2) in order to exploit *inter-SPE* and *intra-SPE* parallelisms, two partition strategies *PSPIB* and *NSNRR* are proposed for parallel XML databases; (3) two parallel algorithms *PPM* and *PPPJ* are proposed for query processing based on the *independent* and *pipelining* parallelisms.

The remainder of this paper is organized as follows. In Section 2, some definitions about node schema, path schema and path instance are given. In Section 3, two data partition strategies are illustrated in detail by some examples. Section 4 gives the experimental results and performance evaluations on the proposed partition strategies. Section 5 concludes this paper and gives future work.

2. Preliminaries

In this section, we informally define some basic concepts used throughout in the paper, including XML data tree, node schema, path schema, path schema expression and path instance.

An XML document is represented in an XML data tree. Figure 1 shows the data tree of a simple document segment extracted from XMark benchmark [8], where *circles* represent element nodes, *ellipses* represent text nodes. Each node has a unique identifier, represented as "&" in front of number in preorder traversal. Edges

Table 1. Path Schemas and Path Instances

Path Schema	Path Instance
/site/open_auctions/open_auction/annotation/author	1. &1->&2->&3->&4->&5 2. &1->&2->&13->&14->&15
/site/open_auctions/open_auction/annotation/description/(parlist/listitem)+/text	1. &1->&2->&3->&4->&6->&7->&8->&9->&10->&11->&12 2. &1->&2->&13->&14->&16->&17->&18->&19->&20

between nodes represent composite relationships.

A node schema of a given node is informally defined as the name of the node. For examples, node *&4* has node schema *annotation* and node *&19* has node schema *text*. If the names of two nodes are same, then their node schemas are said to be identical. Node schema is helpful to exploit intra-SPE parallelism. For example, allotting the nodes with the same node schema to the same site can reduce the access cost of each path step of a SPE, while allotting the nodes with different node schema to different sites in round-robin way can avoid the data skew to some degree. The path instance of a node is informally defined a sequence of identifiers from the root to the node, connected by "->". For examples, the path instance of *&5* is *&1->&2->&3->&4->&5*. Similarly, the path schema of a node is informally defined a sequence of node schemas from the root to the node, connected by "/". For examples, the path schema of *&5* is */site/open_auctions/open_auction/annotation/author*. A path schema expression may be either an absolute PSE (APSE) starting from the root, or a relative PSE (RPSE) starting from an arbitrary element node. The BNF of the path schema expression PSE is shown in Figure 2, where '+' indicates one or more occurrences of a node. Because of the introduction of '+', different path schemas may have the same PSEs. Many path schemas can thus be simplified to have the same PSE. If the PSEs between two path schemas are same, two path schemas are said to be equivalent. It is obvious that there are two path schemas and four path instances in the data tree in Figure 1, shown in Table 1. Note that the path schemas of &11 and &19 are justified to be equivalent due to the existence of "+".

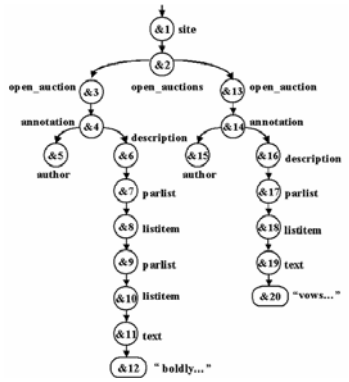

Figure 1. A sample XML data tree

```
PSE ::= APSE | RPSE
APSE ::= '/' RPSE
RPSE ::= RPSE '/' RPSE | RPSE '+' | '(' RPSE ')' | PSESTEP
PSESTEP ::= tag name
tag name ::= string
```

Figure 2. BNF of path schema expression

3. Two partition strategies

In this section, we focus on the data placement of XML data. As discussed in Section 2, the *inter-SPE* and *intra-SPE* parallelisms are two main forms of RPE-based parallelisms. *Path Schema based Path Instance Partition* (PSPIB) physical partition strategy is based on the *inter-SPE* parallelism while *Node Schema based Node Round-Robin* (NSNRR) physical partition strategy on the *intra-SPE* parallelism.

3.1 Path Schema based Path Instance Balancing Strategy

Have discussed in Section 2, *inter-SPE* parallelism is to parallize the executions of SPEs over all sites. Given an RPE query, declustering path instances based on path schema corresponding to a SPE can exploit fully the *inter-SPE* parallelism. Based on this idea, PSPIB strategy is proposed. Its main idea is to decluster all path instances with the same path schema over all sites to minimize response time. Figure 3 shows the partition result of the sample data tree in the *PSPIB* strategy in the

case of two sites, where solid circles represent the placements in site 1 and dotted circles represent the placements in the other site. Shade nodes represent the duplicated nodes in two sites.

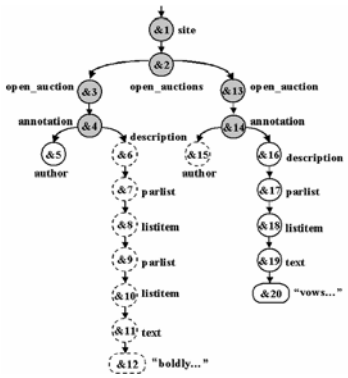

Figure 3. Placements of Data Tree in PSPIB Strategy

PSPIB strategy has some unique properties. (1) Partitioning parallelism or I/O parallelism is fully exploited. (2) It supports the *intra-RPE* and *inter-SPE* parallelisms. Strictly speaking, this *inter-SPE* parallelism is a type of inherent parallelism. That is, once the path schemas related to SPEs are decided, the sites participated in an RPE query are decided accordingly. The *inter-SPE parallelism* is subjected to path schema. (3) By executing several RPE queries simultaneously, the *inter-RPE* parallelism can be achieved so as to maximize the system throughput. (4) There is no data skew in PSPIB strategy due to the constraints of three declustering principles. (5) Due to the existence of redundant nodes, more storage spaces are needed.

3.2 Node Schema based Node Round-Robin Strategy

An RPE can be rewritten to multiple independent SPEs connected only by "/" operators and each SPE can be further divided into multiple path steps. And these multiple path steps are executed in parallel over all sites. The *NSNRR* strategy is proposed based on the *intra-SPE* parallelism. The nodes with identical node schema are clustered to the same site to reduce the access cost of each path step of a SPE effectively, while nodes with different node schema are allotted to different sites in round-robin way to avoid the data skew to some degree. Meanwhile, in each site, the number of nodes should be balanced as evenly as possible. A SPE query is a type of navigational query, which means the navigational accesses are executed from one object to another object through an associated reference between them. This associated reference is represented as "/" in a SPE, which denotes the direct parent-child relationship. When allocating an element node to a certain site, the parent OID information of this element node requires being attached as an attribute. Note that, text nodes are allotted together with its parent element node. Figure 4 shows the partition result of the sample data tree in the *NSNRR* strategy in the case of two sites. Unlike the placements in PSPIB, there is no redundant node. It is easy to see that all object nodes are distributed over two sites according to each node's node schema. That is obvious the nodes with the same node schema are allocated to a site and the nodes with different node schema are assigned to different sites in the round-robin way.

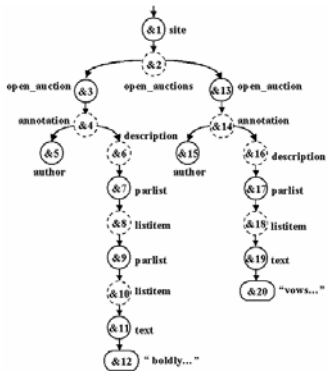

Figure 4. Placements of data tree in NSNRR strategy

The *NSNRR* strategy has the following properties. (1) It supports the *intra-RPE* and *intra-SPE* parallelisms. The *Intra-SPE* parallelism is subjected to node schema and is exploited in the pipelining way based on the right-deep tree scheduling mechanism. (2) Similar to the *PSPIB* strategy, it can also make full use of the *inter-RPE* parallelism to maximize the system throughput. (3) It has lower storage overhead than the *PSPIB* strategy since there is no redundant nodes. (4) There may be data skew in the *NSNRR* strategy. This shortcoming is caused by the random of the round-robin method.

4. Experimental Performance Evaluation

The test environment is based on the network of PCs consisting of 6 sites. All sites have the same configuration of PIII 800HZ CPU, 128MB of memory, 20G hard disk and Solaris 8.0 Intel platform edition of Sun Microsystems operating system and are connected with 100Mbps high-speed LAN. One site acts as a front-end client and the other 5 sites act as parallel servers. The testing programs were coded with C++ and INADA conformed to ODMG C++ binding [9]. We choose the XMark Benchmark to test the performance of the strategies and algorithms proposed in this paper. The test databases consist of 20M, 40M, 60M, 80M and 100M scaling single document.

4.1 Speedup

Speedup holds the problem size constant, and grows the system [10]. Linear speedup, twice as much hardware can perform the task in half the elapsed time, is one of ideal goals of parallel system. The speedup curves of Q5 under two partition strategies were shown from Figure 5(a) to 5(e). It is easy to see that: (1) when document is small, the speedup of NSNRR is better than PSPIB. (2) As the document grows large, the speedup of PSPIB is closer to linear than NSNRR. That is, the performance of PSPIB is better than the NSNRR. In addition, the speedup curve of NSNRR occurs horizontal ladder. This phenomenon is the result of implicit data skew in NSNRR. In other words, it is possible that the object nodes corresponding to a query were allocated to partial sites and only these sites associated with current query were accessed frequently, but other left sites were not accessed at all, despite the number of sites is increased gradually.

4. 2 Scaleup

Scaleup measures the ability to grow both the system and the problem [10]. Both PSPIB and NSNRR scaleup curves of Q5 are shown in Figure 6. It is easy to see that: as the number of sites is increased, the scaleup curve of NSNRR decreases smoothly. On the contrary, the scaleup

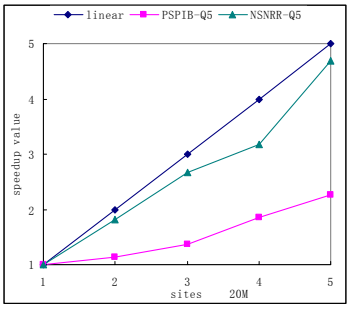

Figure 5(a). The speedup curves of Q5 for 20M document.

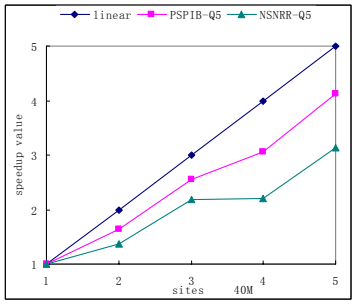

Figure 5(b). The speedup curves of Q5 for 40M document.

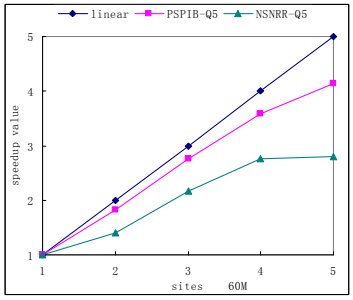

Figure 5(c). The speedup curves of Q5 for 60M document.

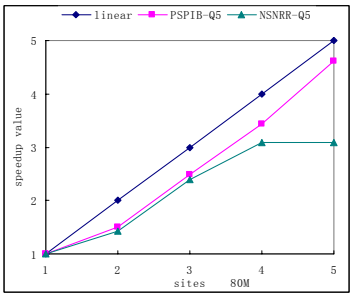

Figure 5(d). The speedup curves of Q5 for 80M document.

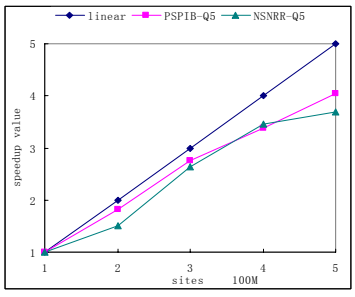

Figure 5(e). The speedup curves of Q5 for 100M document.

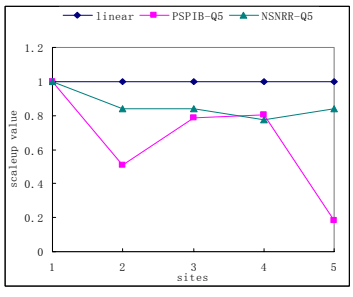

Figure 6. The scaleup curves of Q5.

curve of PSPIB decreases drastically. This proves that NSNRR has the excellent scealeup performance. The scaleup curve of NSNRR is closer to linear scaleup instead the scaleup of PSPIB embodies a vibrate change trend, so the scale up performance of NSNRR is better than PSPIB.

5. Conclusions and Future Work

For the execution of RPE query efficiently and effectively in parallel XML database system, several dramatic kinds of parallelism, intra RPE parallelism, inter-RPE parallelism, intra-SPE parallelism and inter-SPE parallelism are proposed in this paper. We have proposed two partition strategies to exploit intra-RPE parallelism. From the experimental results, we can conclude that: when document is small, the speedup performance of NSNRR strategy is better than PSPIB strategy. Otherwise, when document is large, the speedup performance of PSPIB strategy is better than NSNRR strategy. Further research involves the issues that how to decluster XML data when we take account of the query frequencies of workloads. In this paper, we declustered XML data only based on the static data themselves, without considering the influences of query workloads to partition. But in fact, although data have been declustered as possible as evenly, task skew maybe still occur. Thus, how to redistribute data to avoid task skew becomes an important research issue.

References

[1] Bray T, Paoli J, Sperberg-McQueen C M, *et al*. Extensible markup language (XML) 1.0 (Second Edition) [R]. http://www.w3.org/TR/2000/REC-xml-20001006 , 2000.

[2] Quanzhong Li and Bongki Moon. Indexing and Querying XML Data for Regular Path Expression. In Proceedings of the 27th VLDB Conference, pp 361-370, Roma, Italy, 2001.

[3] Jason McHugh, Jennifer Widom. Query Optimization for XML. In *Proceedings of the 25th VLDB Conference*, pp 315-326, Edinburgh Scotland, 1999.

[4] Don Chamberlin, Jonathan Robie and Daniela Florescu. Quilt: An XML query language for heterogeneous data sources. In *International Workshop on the Web and Databases (WebDB'2000)*, Dallas, TX, May 2000.

[5] James Clark and Steve DeRose. XML Path Language (XPath) version 1.0 w3c recommendation. Technical Report REC-xpath-19991116, World Wide Web Consortium, November 1999.

[6] Don Chamberlin, James Clark, Daniela Florescu, *et al*. XQuery 1.0: An XML Query Language, Technical Report WD-xquery-20010607, World Wide Web Consortium, June 2001.

[7] Ming-Syan Chen, Philip S. Yu and Kun-Lung Wu. Scheduling and Processor Allocation for Parallel Execution of Multi-join Queries. In *Proceedings of 8th International Conference on Data Engineering*, pp 58-67, Tempa Arizona, February 1992.

[8] A.R. Schmidt, F. Waas, M.L. Kersten, *et al*. The XML Benchmark Project. CWI Technical Report INS-R0103. Amsterdam, Netherlands, April 30, 2001.

[9] R.G.G. Cattell and Douglas Barry. Object Database Standard: ODMG 2.0. Morgan Kaufmann Publishers, Inc. San Francisco, California. 1997.

[10] David J. Dewitt and Jim Gray. Parallel Database Systems: The future of High Performance Database Processing. ACM Vol.36, No.6, June 1992.

[11] Y. Zhu, K. J. Lü. An Effective Data Placement Strategy for XML documents. In the proceedings of 18th British National Conference On Databases (LNCS 2097), pp 43-56, Springer-Verlag Berlin Heidelberg, 2001.

[12] K. J. Lü, Y. Zhu, W. Sun, *et al*. Parallel processing XML documents. *Proceedings of the International Database Engineering and Applications Symposium (IDEAS'02)*, 2002.

[13] K. Shoens, A. Luniewski, P. Schwarz, J. Stamos, The Rufus System: Information Organization for Semi-Structured Data. *Proc. of the 25th VLDB Conf.*, Dublin, Aug. 1999.

[14] H. Schoning. Tamino - A DBMS designed for XML. *Proc. of the ICDE Conf.*, Heidelberg, Germany, Apr. 2001.